Amiga Mail

May/June 1990

The Commodore Amiga
Technical Newsletter

Copyright © 1990 Commodore-Amiga, Inc. All rights reserved. Some of the materials used herein heve been reproduced with the permission of the authors indicated. Use or reproduction of such materials shall be in accordance with the authors' instructions.

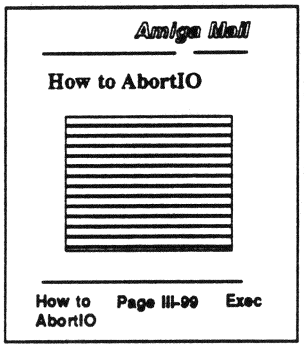
Amiga Mail

Amiga Mail is published six times annually by the Commodore Applications and Technical Support Group (CATS). Our purpose is to support third-party developers of Amiga software and hardware add-ons world-wide by providing technical information and programming guidelines. Each issue features a collection of news and technical articles of special interest to Amiga product developers.

The technical articles in each issue can be saved for future use in the ***Amiga Mail Tech Notes*** *binder, a reference notebook containing all the articles from past issues of Amiga Mail. The Amiga Mail Tech Notes binder is published annually by the CATS group and is available to all developers registered with the Commodore support program.*

For filing technical articles in the Amiga Mail Tech Notes binder, just use the section and page numbers listed at the bottom of each article. The Tech Notes section is shown as a roman numeral followed by the page number within that section.

For instance the article at right would be filed in section III of the Amiga Mail Tech Notes binder, "Exec." The article starts on page 99 of that section in the binder.



For more information on the Amiga Mail Tech Notes binder and the Commodore Developer Support Program, contact the CATS group at (215) 431-9180.

Warning

The information contained herein is subject to change without notice. Commodore specifically does not make any endorsement or representation with respect to the use, results, or performance of the information (including without limitation its capabilities, appropriateness, reliability, currentness or availability).

Disclaimer

This information is provided "as is" without warranty of any kind, either express or implied. The entire risk as to the use of this information is assumed by the user. In no event will Commodore or its affiliated companies be liable for any damages, direct, indirect, incidental, special or consequential, resulting from any defect in the information, even if advised of the possibility of such damages.

Amiga Mail News

Commodore International Announces CDTV at CES

Commodore International unveiled its Commodore Dynamic Total Vision (CDTV) interactive multimedia player June 2nd at the Summer Consumer Electronics Show in Chicago. The player is the first consumer-oriented product to combine the massive storage capabilities of the CD-ROM with the multitasking, audio, graphics, and processing power of an advanced personal computer into a single simple to use, affordable unit. The player connects directly to a television set and home stereo unit to become an interactive information, entertainment, and education center. The unit is controlled by a simple to use infrared remote.

CDTV will have many uses. Through the use of CD-ROM technology, reference materials such as dictionaries, world atlases, and encyclopedias can be accessed and cross referenced at the touch of a button. Packages that teach cooking, sports, and foreign languages as well as those that offer interactive learning for children are planned. Music from rock to opera will be enhanced with computer graphics. Video games will be significantly more sophisticated in depth of play, sound, and graphics than on any other video game or personal computer format. Commodore plans to have a library of more than 100 applications for CDTV available when the product launches. The majority of discs will carry a suggested retail price ranging between $30 and $100.

Nolan Bushnell, general manager of Commodore's new Interactive Consumer Products division, identifies two groups of consumers for CDTV. The first group consists of enthusiasts who currently have advanced audio, video, and computer systems in their homes. The second group is made up of educated adults who want to provide their families with interactive multimedia education, information, and entertainment, but do not want to learn how to use a conventional computer or buy a limited game machine.

IN THIS ISSUE

Using C to Get a Pointer to the CLI Window .....II - 93

Identifying the Amiga's CPU..... III - 25

Collision Detection Between Animation Objects.....V - 25

Stopwatch Functions with the timer.device.....IX - 37

CDTV is the first consumer player to begin to realize the potential of multimedia Compact Disc technology. CD-ROM discs have enormous storage capacity - more than 550 megabytes of data - the equivalent of 250,000 pages of text. This data storage capacity enables program developers to design applications which are significantly more sophisticated, challenging, and more exciting than those available on any existing computer or home entertainment format.

In addition to being able to access data stored on a CD-ROM disc, CDTV can also be used to play conventional audio Compact Discs. In the audio mode, CDTV is fully programmable, and features 8-times oversampling and dual 16-bit D/A converters, a front panel 24-hour clock, stereo headphone jack, and a full-function fluorescent display provides time, track, and volume information at a glance.

The basic CDTV player contains various input and output ports. In addition to the familiar Amiga DB-23 analog RGB video output, the CDTV player has composite video, component video Y-C (S connector type for S-VHS and Hi-8), and RF modulated (F connector) output with optional genlock capabilities via a plug in module. An RS-232 serial interface and a Centronics parallel interface can be found on the unit's rear panel, as well as an external Amiga floppy drive port. Stereo RCA type plugs provide audio out. A front panel personal RAM card port, which allows saving and storing of up to 64K of data, will also be available.

Some additional accessories will include interfaces for an infrared keyboard and an infrared, two-player game controller, a keyboard, an expansion module to house a hard drive, modem, and floppy disk drive, an infrared joystick, and an infrared trackball. There is also a hard-wired alternative to the infrared keyboard, mouse, and joystick.

CDTV is expected to retail for less than $1000. Commodore plans to market the product in the United States through high-end audio/video specialty retailers, upscale department stores, and selected retail chains. Commodore intends to launch CDTV internationally.

Some Additional Specs

- Motorola MC68000 at 7.15909 MHz (NTSC)
- 1 Mb chip RAM
- 2K non-volatile RAM reserved for system (clock, prefs, etc.)
- 512K ROM
- Intelligent video slot (for optional genlock, RF board, etc.) 15 pin edge connector
- DMA slot - for SCSI, LAN, etc.
- Amiga Kickstart 1.3 in ROM
- ISO 9660 File System Handler
- High-speed decompression for graphics, audio, and sound.

Using C to Get a Pointer to the CLI Window

by Rob Wyesham

Programs sometime need to get a pointer to their CLI window. A method for obtaining this pointer was illustrated in the *Amiga Mail* article entitled "CLIColors.asm - Using ConsolePackets in 68000 Assembler" (page II-65, July/August 1989 *Amiga Mail*). However, many C programmers would rather avoid using assembly language. This article shows how to use C to get a pointer the CLI window.

The basic method is the same whether you use C or Assembler. You send an `ACTION_DISK_INFO` packet to the console handler process which includes a pointer to an InfoData struture. When the packet is returned by AmigaDOS, the id_VolumeNode field of the InfoData structure will be filled in with a pointer to the CLI window that the console handler is using.

The program below, WindowPtr.c, follows the code of CLIColors.asm closely. Like the latter program, the strategy behind WindowPtr.c is to:

- ❑ Make sure the program was launched from a CLI.
- ❑ Send an `ACTION_DISK_INFO` packet.
- ❑ Extract the window pointer from the information returned in the packet.

The program starts by examining certain structures to determine if it was launched from a CLI. If it passes that test, the program checks its Process structure to make sure that it has a valid pointer to a console task. Next, the program allocates memory for FindWindow, a custom structure composed of a struct StandardPacket and a struct InfoData.

The program then initializes the StandardPacket fields and sends an ACTION_DISK_INFO packet to the DOS. When the packet returns, the program checks for an error. An error could possibly be caused by the CLI not having an associated window. A CLI that uses the AUX: handler rather than the CON: handler could cause such an error.

If the packet returns without an error, the id_VolumeNode field of the InfoData structure points to the CLI window. This is an ordinary C pointer, not a BPTR. The program shows that the pointer obtained does indeed point to the current CLI window by printing out a few of the window's parameters.

WindowPtr.c is listed below. It was compiled under Lattice 5.05 using the command: LC -b1 -cfist -L -v -w WindowPtr.c. Link with lc.lib and amiga.lib.

```
/*
* File name: windowptr.c
* Purpose: To illustrate how to get the pointer
*            to the current CLI window.
* Requires: Must be started from a CLI
*/

#include <libraries/dosextens.h>        /* Defines the StandardPacket
                                         * structure, the packet types,
                                         * and the Process structure
                                         */

#include <libraries/dos.h>          /* Defines the InfoData structure */
                                    /* and RETURN_XXX                 */
#include <exec/types.h>
#include <exec/memory.h>            /* Defines MEMF_XXX */
#include <stdio.h>                  /* Defines NULL and stdout */
#include <stdlib.h>                 /* Declares exit() */
#include <proto/exec.h>
#include <proto/intuition.h>


/* Set up the FindWindow structure. */
struct FindWindow
{
    struct StandardPacket FW_Pack;      /* A StandardPacket is all long-words */
    struct InfoData   FW_Info;          /* so InfoData is long-word-aligned   */
};                                      /* if the FindWindow struct is.       */


#define THISTASK NULL

extern VOID cleanExit( struct FindWindow *, int );
```

```
VOID main( int argc, char *argv[] )
{
    struct Process *myProcess = NULL;
    struct FindWindow *thisWindow = NULL;
    struct Window *addrThisWindow = NULL;
    SHORT minW, minH;
    USHORT maxW, maxH;
    SHORT xPos, yPos, W, H;

    /* Get address of the current task */
    myProcess = (struct Process *)FindTask( THISTASK );

    /* Make sure this program wasn't started from Workbench */
    if ( argc == 0 ) cleanExit( thisWindow, RETURN_WARN );

    /* Make sure that there is a pr_ConsoleTask */
    if ( myProcess->pr_ConsoleTask == NULL )
       cleanExit( thisWindow, RETURN_WARN );

    /* Allocate memory for the FindWindow structure */
    thisWindow = (struct FindWindow *)
             AllocMem( sizeof( struct FindWindow ), MEMF_PUBLIC|MEMF_CLEAR );
    if ( thisWindow == NULL ) cleanExit( thisWindow, RETURN_WARN );

    /* Initialize the packet */
    thisWindow->FW_Pack.sp_Msg.mn_Node.ln_Name =
                                (char *)&thisWindow->FW_Pack.sp_Pkt;
    thisWindow->FW_Pack.sp_Pkt.dp_Link = (struct Message *)thisWindow;
    thisWindow->FW_Pack.sp_Pkt.dp_Port = &myProcess->pr_MsgPort;
    thisWindow->FW_Pack.sp_Pkt.dp_Type = ((LONG)&thisWindow->FW_Info) >> 2;

    /* Now send the packet, and wait for it to return */
    PutMsg( (struct MsgPort *)myProcess->pr_ConsoleTask,
                                   (struct Message *)thisWindow );

    (void)WaitPort( (struct MsgPort *) &myProcess->pr_MsgPort );

    /* Our program has been replied to, so get the reply message */
    (void)GetMsg( &myProcess->pr_MsgPort );

    /* Save the address of the CLI window */
    addrThisWindow = (struct Window *) (&(thisWindow->FW_Info))->id_VolumeNode;


    /* If there was an error, terminate execution */
    if ( thisWindow->FW_Pack.sp_Pkt.dp_Res1 == DOSFALSE )
             cleanExit( thisWindow, RETURN_WARN );

    if ( addrThisWindow == NULL ) cleanExit( thisWindow, RETURN_WARN );
```

```
/* Get the position, dimensions, and minimum and maximum
     * dimensions of the currently active CLI window */
    xPos = addrThisWindow->LeftEdge;
    yPos = addrThisWindow->TopEdge;
    W = addrThisWindow->Width;
    H = addrThisWindow->Height;
    minW = addrThisWindow->MinWidth;
    minH = addrThisWindow->MinHeight;
    maxW = addrThisWindow->MaxWidth;
    maxH = addrThisWindow->MaxHeight;


    /* Print the position, dimensions, and minimum and maximum
     * dimensions of the currently active CLI window */
    printf( "Position: x = %d, y = %d    ", xPos, yPos );
    printf( "Dimensions: w = %d, h = %d\n", W, H );
    printf( "%d <= width <= %d    ", minW, maxW );
    printf( "%d <= height <= %d\n", minH, maxH );

    cleanExit( thisWindow, RETURN_OK );
}


VOID cleanExit( pktptr, returnValue )
struct FindWindow *pktptr;
int returnValue;
{
    if ( pktptr ) FreeMem( pktptr, (LONG)sizeof( struct FindWindow ) );

    exit( returnValue );
}
```

A

Identifying the Amiga's CPU

by Dave Haynie

Amiga's come in a variety of configurations including models that use the Motorola 68000, the 68020 and now the 68030. While it's not that difficult to figure out which CPU you're using by trying certain instructions and trapping the exception if they break, this isn't necessary. In fact, the Amiga OS sets an ExecBase flag on system startup that can identify which processor (and coprocessor) is installed. The CPU type flags are stored in ExecBase->AttnFlags.

Under V1.3 and earlier versions of the OS, the system can't tell the difference between a 68020 and a 68030, or a 68881 and a 68882. That's no big surprise, since both the 68030 and 68882 were introduced well after the last version of the OS was released. Under the new V2.0 version of the operating system, support for identifying these processors has been added. If you are running under V2.0 of the operating system, you can just look at ExecBase->AttnFlags to find out what processor is installed. The AttnFlags field will be set as follows:

```
/****** V2.0 Bit defines for AttnFlags ************************/

/*  Processors and Co-processors: */
#define AFB_68010  0   /*  also set for 68020  */
#define AFB_68020  1   /*  also set for 68030  */
#define AFB_68030  2   /* New flag under V2.0  */
#define AFB_68040  3   /* New flag under V2.0  */
#define AFB_68881  4   /*  also set for 68882  */
#define AFB_68882  5   /* New flag under V2.0  */

#define AFF_68010  (1L<<0)
#define AFF_68020  (1L<<1)
#define AFF_68030  (1L<<2) /* New flag under V2.0  */
#define AFF_68040  (1L<<3) /* New flag under V2.0  */
#define AFF_68881  (1L<<4)
#define AFF_68882  (1L<<5) /* New flag under V2.0  */
```

However, if you are running under V1.3 or earlier versions of the operating system, this method will not work with the 68030 and 68882 processors. To overcome this problem you can use the code shown below to identify which CPU the system is using under V1.3. There are three functions to link with your code to identify the processor and coprocessor: GetCPUType(), GetFPUType(), and GetMMUType().

ULONG GetCPUType(void)
Returns a number, representing the type of CPU in the system: 68000L, 68010L, 68020L, or 68030L.

ULONG GetFPUType(void)
Returns a number, representing the type of FPU in the system: 0L (no FPU), 68881L, or 68882L.

ULONG GetMMUType(void)
Returns a number, representing the type of MMU in the system: 0L (no MMU), 68851L, 68030L, or 0xFFFFFFFFL (this means an FPU responding to a MMU address).

In order to find out which processor is present, GetCPUType() first checks ExecBase->AttnFlags. Under V1.3 if this is set to AFF_68020, then the processor may be either a 68020 or a 68030. GetCPUType() then checks to see if the processor is really a 68030 by trying to invert the instruction burst enable bit, which doesn't exist on the 68020. If that bit can be changed, then it is a 68030 system.

Similar methods are used in GetMMUType() and GetFPUType(). These functions first look at ExecBase->AttnFlags and then do extra tests based on unique features of a given coprocessor to find out what is present in the system. The functions are listed below.

```
;------------------------------------------------------------------------
;     From SetCPU V1.5
;     by Dave Haynie
;
;     68030 Assembly Function Module
;
;     This module contains functions that access features of the 68020,
;     68030, and 68851 chips, and ID all of these, plus the 68881/68882
;     FPU chips, reset stuff, and exception handler.
;------------------------------------------------------------------------
;     Macros & constants used herein...
;------------------------------------------------------------------------

CALLSYS macro   *
     jsr     LVO\1(A6)
     endm

CIB_ENABLEEQU  0
CIB_FREEZEEQU  1
CIB_ENTRY EQU  2
CIB_CLEAR EQU  3
CIB_BURST EQU  4

CDB_ENABLEEQU  8
CDB_FREEZEEQU  9
CDB_ENTRY EQU  10
CDB_CLEAR EQU  11
CDB_BURST EQU  12
CDB_WALLOCEQU  13

AFB_68030 EQU  2

ATNFLGS        EQU  $129

LVOSupervisor  EQU  -30
LVOSuperState  EQU  -150
LVOFindTask    EQU  -294
LVOAllocTrap   EQU  -342
LVOFreeTrap    EQU  -348
```

```
;---------------------------------------------------------------------
;     Need just a little more stuff
;---------------------------------------------------------------------

      NOLIST
      include "exec/execbase.i"
      include "exec/tasks.i"
      LIST

*     machine mc68020
*           mc68881

;*********************************************************************
;     This section contains functions that identify and operate on CPU
;     things.
;*********************************************************************

      XDEF _GetCPUType     ; ID the CPU
      XDEF _GetFPUType     ; ID the FPU
      XDEF _GetMMUType     ; ID the MMU

;---------------------------------------------------------------------
;     This function returns the type of the CPU in the system as a
;     longword: 68000, 68010, 68020, or 68030.  The testing must be done
;     in reverse order, in that any higher CPU also has the bits set for
;     a lower CPU.  Also, since 1.3 doesn't recognize the 68030, if I
;     find the 68020 bit set, I always check for the presence of a
;     68030.
;
;     This routine should be the first test routine called under 1.2
;     and 1.3.
;
;     ULONG GetCPUType(void);
;---------------------------------------------------------------------

_GetCPUType:
      movem.l   a4/a5,-(sp)               ; Save this register
      move.l    4,a6                      ; Get ExecBase
      btst.b    #AFB_68030,ATNFLGS(a6)    ; Does the OS think an '030 is here?
      beq  0$
      move.l    #68030,d0                 ; Sure does...
      movem.l   (sp)+,a4/a5
      rts
0$
      btst.b    #AFB_68020,ATNFLGS(a6)    ; Maybe a 68020?
      bne  2$
      btst.b    #AFB_68010,ATNFLGS(a6)    ; Maybe a 68010?
      bne  1$
      move.l    #68000,d0                 ; Just a humble '000
      movem.l   (sp)+,a4/a5
      rts
1$
      move.l    #68010,d0                 ; Yup, we're an '010
      movem.l   (sp)+,a4/a5
      rts
2$
      move.l    #68020,d0                 ; Assume we're an '020
      lea  3$,a5                          ; Get the start of the supervisor code
      CALLSYS   Supervisor
      movem.l   (sp)+,a4/a5
      rts
3$
      moveccacr,d1                        ; Get the cache register
      move.l    d1,a4                     ; Save it for a minute
      bset.l    #CIB_BURST,d1             ; Set the inst burst bit
      bclr.l    #CIB_ENABLE,d1            ; Clear the inst cache bit
      movecd1,cacr                        ; Try to set the CACR
      moveccacr,d1
      btst.l    #CIB_BURST,d1             ; Do we have a set burst bit?
      beq  4$
      move.l    #68030,d0                 ; It's a 68030
      bset.b    #AFB_68030,ATNFLGS(a6)
4$
      move.l    a4,d1                     ; Restore the original CACR
      movecd1,cacr
      rte

;---------------------------------------------------------------------
;     This function returns 0L if the system contains no MMU,
;     68851L if the system does contain an 68851, or 68030L if the
;     system contains a 68030 (built-in MMU).
;
;     This code runs just fine on boards from Ronin and
;     Commodore, as well as all 68030 boards it's been tested on.
;
;     ULONG GetMMUType(void)
;---------------------------------------------------------------------
```

```
_GetMMUType:
     move.l    4,a6                    ; Get ExecBase
     movem.l   a3/a4/a5,-(sp)          ; Save this stuff
     move.l    #0,a1
     CALLSYS   FindTask                ; Call FindTask(0L)
     move.l    d0,a3

     move.l    TC_TRAPCODE(a3),a4      ; Change the exception vector
     move.l    #2$,TC_TRAPCODE(a3)

     move.l    #-1,d0                  ; Try to detect undecode FPU
     subq.l    #4,sp                   ; Let's try an MMU instruction
     dc.w $f017                        ; like PMOVE tc,(sp)
     dc.w $4200
     cmpi #0,d0                        ; Any MMU here?
     beq  1$
     cmpi #-1,d0                       ; Hardware "features"?
     beq  1$
     btst.b    #AFB_68030,ATNFLGS(a6)  ; Does the OS think an '030 is here?
     beq  1$
     move.l    #68030,d0

1$
     addq.l    #4,sp                   ; Return that local
     move.l    a4,TC_TRAPCODE(a3)      ; Reset exception stuff
     movem.l   (sp)+,a3/a4/a5          ; and return the registers
     rts

     ; This is the exception code.  No matter what machine we're on,
     ; we get an exception.  If the MMU's in place, we should get a
     ; privilige violation; if not, an F-Line emulation exception.
2$
     move.l    (sp)+,d0                ; Get Amiga supplied exception #
     cmpi #11,d0                       ; Is it an F-Line?
     beq  3$                           ; If so, go to the fail routine
     move.l    #68851,d0               ; We have MMU
     addq.l    #4,2(sp)                ; Skip the MMU instruction
     rte
3$
     moveq.l   #0,d0                   ; It dinna woik,
     addq.l    #4,2(sp)                ; Skip the MMU instruction
     rte

;
;    This function returns the type of the FPU in the system as a
;    longword: 0 (no FPU), 68881, or 68882.
;
;    ULONG GetFPUType(void);
;

_GetFPUType:
     move.l    a5,-(sp)                ; Save this register
     move.l    4,a6                    ; Get ExecBase
     btst.b    #AFB_68881,ATNFLGS(a6)  ; Does the OS think an FPU is here?
     bne  1$
     moveq.l   #0,d0                   ; No FPU here
     move.l    (sp)+,a5                ; Give back the register
     rts
1$
     lea  2$,a5                        ; Get the start of the supervisor code
     CALLSYS   Supervisor
     move.l    (sp)+,a5                ; Give back registers
     rts
2$
     move.l    #68881,d0               ; Assume we're a 68881
     fsave-(sp)                        ; Test and check
     moveq.l   #0,d1
     move.b    1(sp),d1                ; Size of this frame
     cmpi #$18,d1
     beq 3$
     move.l    #68882,d0               ; It's a 68882
3$
     frestore (sp)+                    ; Restore the stack
     rte

     end
```

Collision Detection Between Animation Objects

by Ken Farinsky

Collision detection allows the system to manage Gel-to-Border and Gel-to-Gel collision handling for an application. "Gels" are VSprite-based objects that may be displayed and moved about the Amiga display. These objects include VSprites (Virtual Sprites), Bobs (Blitter Objects), and AnimObs (Animation Objects). For a detailed discussion of collision detection between simple Gels (such as VSprites and Bobs), see the "Graphics: Sprites, Bobs and Animation" chapter in the *ROM Kernel Manual: Libraries and Devices*.

The basic premise of Amiga collision detection is to call a user supplied collision routine when two objects overlap. This is complicated by two further enhancements. First, borders are counted as objects and special-case processing of collisions is provided between Gels and the border. Second, each Gel has two masks that indicate how pairs of objects will react when they collide.

Collision detection between objects such as VSprites and Bobs is conceptually very simple. Each object is built upon a single underlying structure, the VSprite structure. After the Gel has been added to the system, the DoCollision() routine scans the list of VSprites for overlapping objects and calls the appropriate user-supplied collision routine in the event of a collision. Since there is a one-to-one correspondence between VSprites and objects, it is obvious which VSprite structure belongs to which object on the screen.

Animation Objects Are Not Simple Gels

Animation Objects contain multiple VSprite structures which are linked to form a larger object. When the object is being animated, the system cycles through these underlying components, displaying each at the appropriate moment. At any given point in time the system may be in the process of removing the last displayed component and adding in the next one. During such a component switch, the system will have two VSprites added for a single object: one is in the process of being removed and one is in the process of being displayed.

Special Initialization Steps for Complex Gel Collision Detection

When setting-up a multi-component Gel, each underlying VSprite structure must be initialized for collision, as the system will not look to the others for more information. An AnimOb's collision detection information need not be static. The collision information can change from frame to frame in the same manner image data can change. The example shows a special case, where all frames are similar enough that they have identical collision information.

The following fields in the VSprite structure must be properly initialized for system collision detection: HitMask, MeMask, BorderLine and CollMask. HitMask and MeMask tell the system which collision routine to call when a collision is detected. BorderLine and CollMask are "shadow masks" that are used to determine if this component overlaps any other object. These values are set just as they would be for a single VSprite or a Bob.

Note that it is impossible for components of a single AnimOb's sequence to collide with one another. However, in an AnimOb that contains multiple sequences, it is possible for a component in one sequence to collide with a component from a different sequence. This can be prevented through careful selection of HitMask and MeMask values, or can be detected in the individual collision routines.

What Happens When Complex Gels Collide

Collisions between Animation Objects *should* be fairly simple to process. The DoCollision() routine *should* examine all displayed objects for both overlap of objects and for correct interaction between the HitMask and MeMask, then call the appropriate collision routine.

Unfortunately, the DoCollision() routine does not ignore the previous AnimOb component, which is in the process of being removed from the display. These components are moved by the system to a position well above and to the left of the visible display. When the Gels are next drawn, the removed component will be moved off-screen, causing it to disappear from the display. At the same time, the new component will be drawn on-screen. This would not be a problem if the removed components did not overlap, however, all components that are being removed are positioned at the same off-screen location. This means that they all overlap and collisions will occur between all pairs of components that are in the process of being removed.

The BOBSAWAY Flag Comes to the Rescue

To handle collisions between pairs of AnimObs, the Gel-to-Gel collision routines must ignore the objects that are being removed. There are two ways to handle this. The first is to check for large negative position values in the X and Y variables of the VSprite structure. This works, but is not the best way to check for removal. The second, and preferred method, is to look at the flags of the associated Bob. If the BOBSAWAY flag is set, then the bob is being removed and should not be used in the collision routines. Note that this has been changed for 2.0, which will only look at active VSprites for collision detection.

The example code is made up of four files:

1) animtools.h - include file necessary to use animtools.
2) animtools_proto.h - prototypes for animtools functions.
3) animtools.c - animtools functions. These simplify the use of the animation sub-system.
4) collide.c - example code.

The example code was published in the latest release of the *ROM Kernel Manual: Libraries & Devices*. It has been updated and reprinted here. The example can be compiled and linked using Lattice C with these commands:

```
lc -bl -cfist -v -y -ocollide.o collide.c
lc -bl -cfist -v -y -oanimtools.o animtools.c
blink from lib:c.o collide.o animtools.o library
                          lib:lc.lib lib:amiga.lib to collide
```

1) /* File: animtools.h */

```
#ifndef GELTOOLS_H
#define GELTOOLS_H

/* these data structures are used by the functions in animtools.c to
** allow for an easier interface to the animation system.
*/

/* data structure to hold information for a new vsprite.
** note that:
**     NEWVSPRITE myNVS;
** is equivalent to:
**     struct newVSprite myNVS;
*/
typedef struct newVSprite
    {
    WORD           *nvs_Image;      /* image data for the vsprite   */
    WORD           *nvs_ColorSet;   /* color array for the vsprite  */
    SHORT           nvs_WordWidth;  /* width in words               */
    SHORT           nvs_LineHeight; /* height in lines              */
    SHORT           nvs_ImageDepth; /* depth of the image           */
    SHORT           nvs_X;          /* initial x position           */
    SHORT           nvs_Y;          /* initial y position           */
    SHORT           nvs_Flags;      /* vsprite flags                */

/* THIS HAS CHANGED for amiga mail. **********************************
**
** Add two lines to the NEWVSPRITE structure.
** This allows the created VSprite structures to be initialized with
** HitMask and MeMask information.
*/
    USHORT          nvs_HitMask;    /* Hit mask.                    */
    USHORT          nvs_MeMask;     /* Me mask.                     */
/*
** END OF CHANGE for amiga mail.    **********************************
*/
    } NEWVSPRITE;

/* data structure to hold information for a new bob.
** note that:
**     NEWBOB myNBob;
** is equivalent to:
**     struct newBob myNBob;
*/
typedef struct newBob
    {
    WORD       *nb_Image;       /* image data for the bob       */
    SHORT       nb_WordWidth;   /* width in words               */
    SHORT       nb_LineHeight;  /* height in lines              */
    SHORT       nb_ImageDepth;  /* depth of the image           */
    SHORT       nb_PlanePick;   /* planes that get image data   */
    SHORT       nb_PlaneOnOff;  /* unused planes to turn on     */
    SHORT       nb_BFlags;      /* bob flags                    */
    SHORT       nb_DBuf;        /* 1=double buf, 0=not          */
    SHORT       nb_RasDepth;    /* depth of the raster          */
    SHORT       nb_X;           /* initial x position           */
    SHORT       nb_Y;           /* initial y position           */
```

```
/* THIS HAS CHANGED for amiga mail. **********************************
**
** Add two lines to the NEWBOB structure.
** This is only used to pass the information to the VSprite structure
** in the makeBob() routine.
*/
    USHORT      nb_HitMask;     /* Hit mask.                    */
    USHORT      nb_MeMask;      /* Me mask.                     */
/*
** END OF CHANGE for amiga mail.    **********************************
*/
    } NEWBOB ;

/* data structure to hold information for a new animation component.
** note that:
**     NEWANIMCOMP myNAC;
** is equivalent to:
**     struct newAnimComp myNAC;
*/
typedef struct newAnimComp
    {
    WORD  (*nac_Routine)(); /* routine called when Comp is displayed.   */
    SHORT   nac_Xt;         /* initial delta offset position.           */
    SHORT   nac_Yt;         /* initial delta offset position.           */
    SHORT   nac_Time;       /* Initial Timer value.                     */
    SHORT   nac_CFlags;     /* Flags for the Component.                 */
    } NEWANIMCOMP;

/* data structure to hold information for a new animation sequence.
** note that:
**     NEWANIMSEQ myNAS;
** is equivalent to:
**     struct newAnimSeq myNAS;
*/
typedef struct newAnimSeq
    {
    struct AnimOb  *nas_HeadOb; /* common Head of Object.               */
    WORD   *nas_Images;         /* array of Comp image data             */
    SHORT  *nas_Xt;             /* arrays of initial offsets.           */
    SHORT  *nas_Yt;             /* arrays of initial offsets.           */
    SHORT  *nas_Times;          /* array of Initial Timer value.        */
    WORD (**nas_Routines)();    /* Array of fns called when comp drawn  */
    SHORT   nas_CFlags;         /* Flags for the Component.             */
    SHORT   nas_Count;          /* Num Comps in seq (= arrays size)     */

/* THIS HAS CHANGED for amiga mail. **********************************
**
** Delete two lines from the NEWANIMSEQ structure.
** These are removed becaues the information is now contained in the
** NEWBOB and NEWVSPRITE structures.
**
**  SHORT   nas_HitMask;
**  SHORT   nas_MeMask;
**
** END OF CHANGE for amiga mail.    **********************************
*/
    SHORT   nas_SingleImage;    /* one (or count) images.               */
    } NEWANIMSEQ;

#endif
```

2) /* File: animtools_proto.h */

```
struct GelsInfo *setupGelSys(struct RastPort *rPort, BYTE reserved);
VOID cleanupGelSys(struct GelsInfo *gInfo, struct RastPort *rPort);

struct VSprite *makeVSprite(NEWVSPRITE *nVSprite);
struct Bob *makeBob(NEWBOB *nBob);
struct AnimComp *makeComp(NEWBOB *nBob, NEWANIMCOMP *nAnimComp);
struct AnimComp *makeSeq(NEWBOB *nBob, NEWANIMSEQ *nAnimSeq);

VOID freeVSprite(struct VSprite *vsprite);
VOID freeBob(struct Bob *bob, LONG rasdepth);
VOID freeComp(struct AnimComp *myComp, LONG rasdepth);
VOID freeSeq(struct AnimComp *headComp, LONG rasdepth);
VOID freeOb(struct AnimOb *headOb, LONG rasdepth);
```

3) /* **animtools.c 19oct89** original code by Dave Lucas.

```
** rework by (CATS)
**
** This file is a collection of tools which are used with the VSprite, Bob
** and Animation system software. It is intended as a useful EXAMPLE, and
** while it shows what must be done, it is not the only way to do it.
** If Not Enough Memory, or error return, each cleans up after itself
** before returning.
**
** NOTE:  these routines assume a very specific structure to the
** gel lists.  make sure that you use the correct pairs together
** (i.e. makeOb()/freeOb(), etc.)
**
** lattice c 5.04
** lc -bl -cfist -v -y -oanimtools.o animtools.c
*/

#include <exec/types.h>
#include <exec/memory.h>
#include <graphics/gfx.h>
#include <graphics/gels.h>
#include <graphics/clip.h>
#include <graphics/rastport.h>
#include <graphics/view.h>
#include <graphics/gfxbase.h>

/* THIS HAS CHANGED for amiga mail. **********************************
**
** In the RKM it was assumed that animtools stuff was in a different
** directory from the example program.
**
**      #include "/animtools/animtools.h"
**      #include "/animtools/animtools_proto.h"
**
** Now they are in the same directory as the example:
**
** END OF CHANGE for amiga mail.    **********************************
*/
#include "animtools.h"
#include "animtools_proto.h"

#include <proto/all.h>
```

```
/*-----------------------------------------------------------
** setup the gels system.  After this call is made you can use
** vsprites, bobs, anim comps, and anim obs.
**
** note that this links the GelsInfo structure into the rast port,
** and calls InitGels().
**
** all resources are properly freed on failure.
**
** It uses information in your RastPort structure to establish
** boundary collision defaults at the outer edges of the raster.
**
** This routine sets up for everything - collision detection and all.
**
** You must already have run LoadView before ReadyGelSys is called.
*/
struct GelsInfo *setupGelSys(struct RastPort *rPort, BYTE reserved)
{
struct GelsInfo *gInfo;
struct VSprite  *vsHead;
struct VSprite  *vsTail;

if (NULL != (gInfo =
     (struct GelsInfo *)AllocMem((LONG)sizeof(struct GelsInfo), MEMF_CLEAR)))
     {
     if (NULL != (gInfo->nextLine =
          (WORD *)AllocMem((LONG)sizeof(WORD) * 8, MEMF_CLEAR)))
          {
          if (NULL != (gInfo->lastColor =
               (WORD **)AllocMem((LONG)sizeof(LONG) * 8, MEMF_CLEAR)))
               {
               if (NULL != (gInfo->collHandler =
                    (struct collTable *)AllocMem((LONG)sizeof(struct
                                                          collTable), MEMF_CLEAR)))
                    {
                    if (NULL != (vsHead = (struct VSprite *)AllocMem(
                         (LONG)sizeof(struct VSprite), MEMF_CLEAR)))
                         {
                         if (NULL != (vsTail = (struct VSprite *)AllocMem(
                              (LONG)sizeof(struct VSprite), MEMF_CLEAR)))
                              {
                              gInfo->sprRsrvd   = reserved;

/* THIS HAS CHANGED for amiga mail. **********************************
**
** In the RKM leftmost and topmost were set to zero.
**
**                       gInfo->leftmost   = 0;
**                       gInfo->topmost    = 0;
**
** This has been changed to one to better keep items inside the
** boundaries of the display.
*/
                              gInfo->leftmost   = 1;
                              gInfo->topmost    = 1;
/*
** END OF CHANGE for amiga mail.    **********************************
*/

                              gInfo->rightmost  =
                                  (rPort->BitMap->BytesPerRow << 3) - 1;
                              gInfo->bottommost = rPort->BitMap->Rows - 1;

                              rPort->GelsInfo = gInfo;
```

```
                    InitGels(vsHead, vsTail, gInfo);

                    return(gInfo);
                    }
                FreeMem(vsHead, (LONG)sizeof(*vsHead));
                }
            FreeMem(gInfo->collHandler,
                                (LONG)sizeof(struct collTable));
            }
        FreeMem(gInfo->lastColor, (LONG)sizeof(LONG) * 8);
        }
    FreeMem(gInfo->nextLine, (LONG)sizeof(WORD) * 8);
    }
FreeMem(gInfo, (LONG)sizeof(*gInfo));
}
return(NULL);
}

/*-----------------------------------------------------------
** free all of the stuff allocated by setupGelSys().
** only call this routine if setupGelSys() returned successfully.
** the GelsInfo structure is the one returned by setupGelSys().
**
** It also unlinks the GelsInfo from the RastPort.
*/
VOID cleanupGelSys(struct GelsInfo *gInfo, struct RastPort *rPort)
{
rPort->GelsInfo = NULL;

FreeMem(gInfo->collHandler, (LONG)sizeof(struct collTable));
FreeMem(gInfo->lastColor, (LONG)sizeof(LONG) * 8);
FreeMem(gInfo->nextLine, (LONG)sizeof(WORD) * 8);
FreeMem(gInfo->gelHead, (LONG)sizeof(struct VSprite));
FreeMem(gInfo->gelTail, (LONG)sizeof(struct VSprite));
FreeMem(gInfo, (LONG)sizeof(*gInfo));
}

/*-----------------------------------------------------------
** create a VSprite from the information given in nVSprite.
** use freeVSprite() to free this gel.
*/
struct VSprite *makeVSprite(NEWVSPRITE *nVSprite)
{
struct VSprite *vsprite;
LONG            line_size;
LONG            plane_size;

line_size = (LONG)sizeof(WORD) * nVSprite->nvs_WordWidth;
plane_size = line_size * nVSprite->nvs_LineHeight;

if (NULL != (vsprite =
    (struct VSprite *)AllocMem((LONG)sizeof(struct VSprite), MEMF_CLEAR)))
    {
    if (NULL != (vsprite->BorderLine =
        (WORD *)AllocMem(line_size, MEMF_CHIP)))
        {
        if (NULL != (vsprite->CollMask =
            (WORD *)AllocMem(plane_size, MEMF_CHIP)))
            {
            vsprite->Y          = nVSprite->nvs_Y;
            vsprite->X          = nVSprite->nvs_X;
            vsprite->Flags      = nVSprite->nvs_Flags;
            vsprite->Width      = nVSprite->nvs_WordWidth;
```

```
            vsprite->Depth      = nVSprite->nvs_ImageDepth;
            vsprite->Height     = nVSprite->nvs_LineHeight;

/* THIS HAS CHANGED for amiga mail. **********************************
**
** In the RKM, the vsprite HitMask and MeMask were set to an arbitrary
** value (it was assumed that the programmer would later patch these
** values if collision detection was required!)
**
**          vsprite->MeMask     = 1;
**          vsprite->HitMask    = 1;
**
** Here, we have changed the NEWVSPRITE structure to contain these
** values, so this routine can set-up both of them.
*/
            vsprite->MeMask     = nVSprite->nvs_MeMask;
            vsprite->HitMask    = nVSprite->nvs_HitMask;
/*
** END OF CHANGE for amiga mail.    **********************************
*/

            vsprite->ImageData  = nVSprite->nvs_Image;
            vsprite->SprColors  = nVSprite->nvs_ColorSet;
            vsprite->PlanePick  = 0x00;
            vsprite->PlaneOnOff = 0x00;

            InitMasks(vsprite);
            return(vsprite);
            }
        FreeMem(vsprite->BorderLine, line_size);
        }
    FreeMem(vsprite, (LONG)sizeof(*vsprite));
    }
return(NULL);
}

/*-----------------------------------------------------------------
** create a Bob from the information given in nBob.
** use freeBob() to free this gel.
**
** A VSprite is created for this bob.
** This routine properly allocates all double buffered information
** if it is required.
*/
struct Bob *makeBob(NEWBOB *nBob)
{
struct Bob         *bob;
struct VSprite     *vsprite;
NEWVSPRITE          nVSprite ;
LONG                rassize;

rassize = (LONG)sizeof(UWORD) *
            nBob->nb_WordWidth * nBob->nb_LineHeight * nBob->nb_RasDepth;

if (NULL != (bob =
    (struct Bob *)AllocMem((LONG)sizeof(struct Bob), MEMF_CLEAR)))
    {
    if (NULL != (bob->SaveBuffer = (WORD *)AllocMem(rassize, MEMF_CHIP)))
        {
        nVSprite.nvs_WordWidth  = nBob->nb_WordWidth;
        nVSprite.nvs_LineHeight = nBob->nb_LineHeight;
        nVSprite.nvs_ImageDepth = nBob->nb_ImageDepth;
        nVSprite.nvs_Image      = nBob->nb_Image;
        nVSprite.nvs_X          = nBob->nb_X;
```

```
        nVSprite.nvs_Y          = nBob->nb_Y;
        nVSprite.nvs_ColorSet   = NULL;
        nVSprite.nvs_Flags      = nBob->nb_BFlags;

/* THIS HAS CHANGED for amiga mail. *********************************
**
** The structure of the program has been changed to pass the values
** of the HitMask and the MeMask down to the VSprite structure.
** Push the values into the NEWVSPRITE structure for use in
** makeVSprite().
*/
        nVSprite.nvs_MeMask     = nBob->nb_MeMask;
        nVSprite.nvs_HitMask    = nBob->nb_HitMask;
/*
** END OF CHANGE for amiga mail.    *********************************
*/

        if ((vsprite = makeVSprite(&nVSprite)) != NULL)
            {
            vsprite->PlanePick = nBob->nb_PlanePick;
            vsprite->PlaneOnOff = nBob->nb_PlaneOnOff;

            vsprite->VSBob   = bob;
            bob->BobVSprite  = vsprite;
            bob->ImageShadow = vsprite->CollMask;
            bob->Flags       = 0;
            bob->Before      = NULL;
            bob->After       = NULL;
            bob->BobComp     = NULL;

            if (nBob->nb_DBuf)
                {
                if (NULL != (bob->DBuffer = (struct DBufPacket *)AllocMem(
                    (LONG)sizeof(struct DBufPacket), MEMF_CLEAR)))
                    {
                    if (NULL != (bob->DBuffer->BufBuffer =
                        (WORD *)AllocMem(rassize, MEMF_CHIP)))
                        {
                        return(bob);
                        }
                    FreeMem(bob->DBuffer,
                                (LONG)sizeof(struct DBufPacket));
                    }
                }
            else
                {
                bob->DBuffer = NULL;
                return(bob);
                }

            freeVSprite(vsprite);
            }
        FreeMem(bob->SaveBuffer, rassize);
        }
    FreeMem(bob, (LONG)sizeof(*bob));
    }
return(NULL);
}
```

```
/*-------------------------------------------------------------
** create a Animation Component from the information given in nAnimComp
** and nBob.
** use freeComp() to free this gel.
**
** makeComp calls makeBob(), and links the bob into a AnimComp.
*/
struct AnimComp *makeComp(NEWBOB *nBob, NEWANIMCOMP *nAnimComp)
{
struct Bob      *compBob;
struct AnimComp *aComp;

if ((aComp = AllocMem((LONG)sizeof(struct AnimComp),MEMF_CLEAR)) != NULL)
    {
    if ((compBob = makeBob(nBob)) != NULL)
        {
        compBob->After   = NULL;  /* Caller can deal with these later. */
        compBob->Before  = NULL;
        compBob->BobComp = aComp;   /* Link 'em up. */

        aComp->AnimBob      = compBob;
        aComp->TimeSet      = nAnimComp->nac_Time; /* Num ticks active. */
        aComp->YTrans       = nAnimComp->nac_Yt; /* Offset rel to HeadOb */
        aComp->XTrans       = nAnimComp->nac_Xt;
        aComp->AnimCRoutine = nAnimComp->nac_Routine;
        aComp->Flags        = nAnimComp->nac_CFlags;
        aComp->Timer        = 0;
        aComp->NextSeq      = NULL;
        aComp->PrevSeq      = NULL;
        aComp->NextComp     = NULL;
        aComp->PrevComp     = NULL;
        aComp->HeadOb       = NULL;

        return(aComp);
        }
    FreeMem(aComp, (LONG)sizeof(struct AnimComp));
    }
return(NULL);
}

/*-------------------------------------------------------------
** create an Animation Sequence from the information given in nAnimSeq
** and nBob.
** use freeSeq() to free this gel.
**
** this routine creates a linked list of animation components which
** make up the animation sequence.
**
** It links them all up, making a circular list of the PrevSeq
** and NextSeq pointers. That is to say, the first component of the
** sequences' PrevSeq points to the last component; the last component of
** the sequences' NextSeq points back to the first component.
**
** If dbuf is on, the underlying Bobs'll be set up for double buffering.
** If singleImage is non-zero, the pImages pointer is assumed to point to
** an array of only one image, instead of an array of 'count' images, and
** all Bobs will use the same image.
*/
struct AnimComp *makeSeq(NEWBOB *nBob, NEWANIMSEQ *nAnimSeq)
{
int seq;
struct AnimComp *firstCompInSeq = NULL;
struct AnimComp *seqComp = NULL;
struct AnimComp *lastCompMade = NULL;
```

```
LONG image_size;
NEWANIMCOMP nAnimComp;

/* get the initial image.  this is the only image that is used
** if nAnimSeq->nas_SingleImage is non-zero.
*/
nBob->nb_Image = nAnimSeq->nas_Images;
image_size = nBob->nb_LineHeight * nBob->nb_ImageDepth * nBob->nb_WordWidth;

/* for each comp in the sequence */
for (seq = 0; seq < nAnimSeq->nas_Count; seq++)
    {
    nAnimComp.nac_Xt        = *(nAnimSeq->nas_Xt + seq);
    nAnimComp.nac_Yt        = *(nAnimSeq->nas_Yt + seq);
    nAnimComp.nac_Time      = *(nAnimSeq->nas_Times + seq);
    nAnimComp.nac_Routine   = nAnimSeq->nas_Routines[seq];
    nAnimComp.nac_CFlags    = nAnimSeq->nas_CFlags;

    if ((seqComp = makeComp(nBob, &nAnimComp)) == NULL)
        {
        if (firstCompInSeq != NULL)
            freeSeq(firstCompInSeq, (LONG)nBob->nb_RasDepth);
        return(NULL);
        }

/* THIS HAS CHANGED for amiga mail. **********************************
**
** Remove these two lines from the RKM.  This is because we are now
** passing the HitMask and MeMask directly down to the VSprite structure.
** NOTE that the deleted technique only sets up the two values for one
** VSprite in the AnimComp.  The new technique (used in this example)
** correctly does the set-up of all the VSprites in the AnimComp.
**
**  seqComp->AnimBob->BobVSprite->HitMask = nAnimSeq->nas_HitMask;
**  seqComp->AnimBob->BobVSprite->MeMask = nAnimSeq->nas_MeMask;
**
** END OF CHANGE for amiga mail.    **********************************
*/
    seqComp->HeadOb = nAnimSeq->nas_HeadOb;

    /* Make a note of where the first component is. */
    if (firstCompInSeq == NULL)
        firstCompInSeq = seqComp;

    /* link the component into the list */
    if (lastCompMade != NULL)
        lastCompMade->NextSeq = seqComp;

    seqComp->NextSeq = NULL;
    seqComp->PrevSeq = lastCompMade;
    lastCompMade = seqComp;

    /* If nAnimSeq->nas_SingleImage is zero,
    ** the image array has nAnimSeq->nas_Count images.
    */
    if (!nAnimSeq->nas_SingleImage)
        nBob->nb_Image += image_size;
    }
/* On The last component in the sequence, set Next/Prev to make
** the linked list a loop of components.
*/
lastCompMade->NextSeq = firstCompInSeq;
firstCompInSeq->PrevSeq = lastCompMade;

return(firstCompInSeq);
}
```

```
/*-------------------------------------------------------------
** free the data created by makeVSprite()
**
** assumes images deallocated elsewhere.
*/
VOID freeVSprite(struct VSprite *vsprite)
{
LONG    line_size;
LONG    plane_size;

line_size = (LONG)sizeof(WORD) * vsprite->Width;
plane_size = line_size * vsprite->Height;

FreeMem(vsprite->BorderLine, line_size);
FreeMem(vsprite->CollMask, plane_size);

FreeMem(vsprite, (LONG)sizeof(*vsprite));
}

/*-------------------------------------------------------------
** free the data created by makeBob()
**
** it's important that rasdepth match the depth you
** passed to makeBob() when this gel was made.
** assumes images deallocated elsewhere.
*/
VOID freeBob(struct Bob *bob, LONG rasdepth)
{
LONG    rassize;

rassize =  (LONG)sizeof(UWORD) *
        bob->BobVSprite->Width * bob->BobVSprite->Height * rasdepth;

if (bob->DBuffer != NULL)
    {
    FreeMem(bob->DBuffer->BufBuffer, rassize);
    FreeMem(bob->DBuffer, (LONG)sizeof(struct DBufPacket));
    }
FreeMem(bob->SaveBuffer, rassize);
freeVSprite(bob->BobVSprite);
FreeMem(bob, (LONG)sizeof(*bob));
}

/*-------------------------------------------------------------
** free the data created by makeComp()
**
** it's important that rasdepth match the depth you
** passed to makeComp() when this gel was made.
** assumes images deallocated elsewhere.
*/
VOID freeComp(struct AnimComp *myComp, LONG rasdepth)
{
freeBob(myComp->AnimBob, rasdepth);
FreeMem(myComp, (LONG)sizeof(struct AnimComp));
}

/*-------------------------------------------------------------
** free the data created by makeSeq()
**
** Complimentary to makeSeq(), this routine goes through the NextSeq
** pointers and frees the Components
**
** This routine only goes forward through the list, and so
** it must be passed the first component in the sequence, or the sequence
```

```
** must be circular (which is guaranteed if you use makeSeq()).
**
** it's important that rasdepth match the depth you
** passed to makeSeq() when this gel was made.
** assumes images deallocated elsewhere.
*/
VOID freeSeq(struct AnimComp *headComp, LONG rasdepth)
{
struct AnimComp *curComp;
struct AnimComp *nextComp;

/* this is freeing a loop of AnimComps, hooked together by the
** NextSeq and PrevSeq pointers.
*/

/* break the NextSeq loop, so we get a NULL at the end of the list. */
headComp->PrevSeq->NextSeq = NULL;

curComp = headComp;          /* get the start of the list */
while (curComp != NULL)
     {
     nextComp = curComp->NextSeq;
     freeComp(curComp, rasdepth);
     curComp = nextComp;
     }
}

/*-------------------------------------------------------------
** free an animation object (list of sequences).
**
** freeOb() goes through the NextComp pointers, starting at the AnimObs'
** HeadComp, and frees every sequence.
** it only goes forward. It then frees the Object itself.
** assumes images deallocated elsewhere.
*/
VOID freeOb(struct AnimOb *headOb, LONG rasdepth)
{
struct AnimComp *curSeq;
struct AnimComp *nextSeq;

curSeq = headOb->HeadComp;          /* get the start of the list */
while (curSeq != NULL)
     {
     nextSeq = curSeq->NextComp;
     freeSeq(curSeq, rasdepth);
     curSeq = nextSeq;
     }

FreeMem(headOb, (LONG)sizeof(struct AnimOb));
}
```

```
4) /* Collide.c */
/* This is an example that shows how to do collision detection between
** multiple animation objects and between animation objects and the border.
**
** Lattice C 5.04
** lc -bl -cfist -v -y -ocollide.o collide.c
** lc -bl -cfist -v -y -oanimtools.o animtools.c
** blink from lib:c.o collide.o animtools.o
**       library lib:lc.lib lib:amiga.lib to collide
*/
#include <exec/types.h>
#include <intuition/intuition.h>
#include <graphics/gels.h>
#include <graphics/collide.h>
#include <exec/memory.h>
#include <libraries/dos.h>

#include "animtools.h"
#include "animtools_proto.h"

#include <stdlib.h>
#include <stdio.h>
#include <proto/all.h>
int CXBRK(void) { return(0); }

/* prototypes for functions in this file */
VOID __stdargs __saveds hit_routine(struct VSprite *vs1,struct VSprite *vs2);
VOID __stdargs __saveds bounceWall(struct VSprite *vs1,LONG borderflags);
struct AnimOb *setupBoing(SHORT dbufing);
VOID runAnimation(struct Window *win, SHORT dbufing,
                  struct AnimOb **animKey, struct BitMap **myBitMaps);
LONG setupPlanes(struct BitMap *bitMap, LONG depth, LONG width, LONG height);
struct BitMap **setupBitMaps(LONG depth, LONG width, LONG height);
VOID freePlanes(struct BitMap *bitMap, LONG depth, LONG width, LONG height);
VOID freeBitMaps(struct BitMap **myBitMaps,
                 LONG depth, LONG width, LONG height);
struct GelsInfo *setupDisplay(struct Window **win, SHORT dbufing,
                              struct BitMap **myBitMaps);
VOID drawGels(struct Window *win, struct AnimOb **animKey,
              SHORT dbufing, WORD *toggleFrame, struct BitMap **myBitMaps);

#define RBMWIDTH  320
#define RBMHEIGHT 200
#define RBMDEPTH    4

struct NewScreen ns =
    {
    0, 0, 320, 200, 2, 0, 1, NULL,
    CUSTOMSCREEN, NULL, "Collision With AnimObs", NULL, NULL
    };
struct NewWindow nw =
    {
    50, 50, 220, 100, -1, -1, CLOSEWINDOW,
    WINDOWCLOSE | RMBTRAP, NULL, NULL, "Close Window to Stop", NULL,
    NULL, 150, 100, 150, 100, CUSTOMSCREEN
    };

struct IntuitionBase *IntuitionBase = NULL;
struct GfxBase       *GfxBase       = NULL;

int return_code;

/* these give the number of frames (COUNT), size (HEIGHT, WIDTH, DEPTH),
** and word width (WWIDTH) of the animated object.
*/
```

```
#define BOING_COUNT     6
#define BOING_HEIGHT    25
#define BOING_WIDTH     32
#define BOING_DEPTH      1
#define BOING_WWIDTH    ((BOING_WIDTH + 15) / 16)

/* these are the IDs for the system to use for the three objects.
** these numbers will be used for the collision detection system.
**
** Do not use zero (0), as it is reserved by the collision system
** for border hits (see graphics/collide.h, BORDERHIT.)
*/
#define BOING_1     2
#define BOING_2     3
#define BOING_3     4

/* these are the number of counts that each frame is displayed.
** they are all one, so each frame is displayed once then the
** animation system will move on to the next in the sequence
*/
SHORT boing3Times[BOING_COUNT] = { 1, 1, 1, 1, 1, 1 };

/* these are all set to zero as we do not want anything added
** to the X and Y positions using ring motion control.
** all movement is done using the acceleration and velocity
** values.
*/
SHORT boing3YTranses[BOING_COUNT] = { 0, 0, 0, 0, 0, 0 };
SHORT boing3XTranses[BOING_COUNT] = { 0, 0, 0, 0, 0, 0 };

/* no special routines to call when each anim comp is displayed */
WORD (*boing3CRoutines[BOING_COUNT])(struct AnimComp *) =
    { NULL, NULL, NULL, NULL, NULL, NULL };

UWORD __chip boing3Image[BOING_COUNT][BOING_WWIDTH * BOING_HEIGHT * BOING_DEPTH] =
    {
    /*----- bitmap Boing, frame 0:  w = 32, h = 25 ------ */
        {
        0x0023, 0x0000,  0x004E, 0x3000,  0x00E3, 0x3A00,  0x03C3, 0xC900,
        0x0787, 0x8780,  0x108F, 0x8700,  0x31F7, 0x8790,  0x61F0, 0x4790,
        0x63E0, 0xFB90,  0x43E0, 0xF848,  0x3BC0, 0xF870,  0x3801, 0xF870,
        0x383D, 0xF070,  0x387E, 0x1070,  0x387C, 0x0EE0,  0xD87C, 0x1F10,
        0x467C, 0x1E10,  0x479C, 0x1E30,  0x6787, 0x3E20,  0x0787, 0xCC60,
        0x0F0F, 0x8700,  0x048F, 0x0E00,  0x0277, 0x1C00,  0x0161, 0xD800,
        0x0027, 0x2000,
        },
    /*----- bitmap Boing, frame 1:  w = 32, h = 25 ------ */
        {
        0x0031, 0x8000,  0x0107, 0x1800,  0x00F0, 0x1900,  0x09E1, 0xEC80,
        0x13C1, 0xE340,  0x1803, 0xE380,  0x387B, 0xC390,  0x30F8, 0x01D0,
        0x70F8, 0x3DC0,  0xE1F0, 0x3E08,  0x9DF0, 0x7C30,  0x9E30, 0x7C30,
        0x9E1C, 0x7C30,  0x1C1F, 0x9C30,  0x1C1F, 0x0630,  0x7C1F, 0x0780,
        0x623F, 0x0798,  0x63DE, 0x0F10,  0x23C1, 0x0F20,  0x33C3, 0xEE20,
        0x0BC3, 0xC380,  0x0647, 0xC700,  0x023F, 0x8E00,  0x0130, 0xF800,
        0x0033, 0x8000,
        },
    /*----- bitmap Boing, frame 2:  w = 32, h = 25 ------ */
        {
        0x0019, 0xC000,  0x0103, 0x8800,  0x0278, 0x8D00,  0x0CF0, 0xFE80,
        0x11F0, 0xF140,  0x0E60, 0xF1E0,  0x1C39, 0xF0C0,  0x1C3E, 0x30C0,
        0x387E, 0x0CE0,  0xF87C, 0x1F28,  0x8C7C, 0x1F18,  0x8F3C, 0x1F18,
        0x8F06, 0x1E18,  0x8F07, 0xDE18,  0x8F07, 0xC018,  0x6E0F, 0xC1C0,
        0x300F, 0x83C8,  0x31EF, 0x8390,  0x31F0, 0x8780,  0x11E0, 0xF720,
        0x11E1, 0xF1C0,  0x0B61, 0xE300,  0x071B, 0xC600,  0x0138, 0x6C00,
        0x0031, 0x8000,
        },
```

```
    /*----- bitmap Boing, frame 3:  w = 32, h = 25 ------ */
        {
        0x001C, 0xE000,  0x01B1, 0xCC00,  0x031C, 0xC500,  0x0C3C, 0x3680,
        0x1878, 0x7840,  0x2F70, 0x78E0,  0x0E08, 0x7860,  0x1E0F, 0xB860,
        0x1C1F, 0x0460,  0xBC1F, 0x07B0,  0xC43F, 0x0788,  0xC7FE, 0x0788,
        0xC7C0, 0x0F88,  0xC781, 0xEF88,  0xC783, 0xF118,  0x2783, 0xE0E8,
        0x3983, 0xE1E0,  0x3863, 0xE1C0,  0x1878, 0xC1C0,  0x3878, 0x3380,
        0x10F0, 0x78C0,  0x0B70, 0xF180,  0x0588, 0xE200,  0x009E, 0x2400,
        0x0018, 0xC000,
        },
    /*----- bitmap Boing, frame 4:  w = 32, h = 25 ------ */
        {
        0x000E, 0x6000,  0x00F8, 0xE400,  0x030F, 0xE600,  0x061E, 0x1300,
        0x0C3E, 0x1C80,  0x27FC, 0x1C60,  0x0784, 0x3C60,  0x4F07, 0xFE20,
        0x8F07, 0xC230,  0x1E0F, 0xC1F0,  0x620F, 0x83C8,  0x61CF, 0x83C8,
        0x61E1, 0x83C8,  0x63E0, 0x63C8,  0x63E0, 0xF9C8,  0x03E0, 0xF878,
        0x1DC0, 0xF860,  0x1C21, 0xF0E0,  0x5C3E, 0xF0C0,  0x0C3C, 0x11C0,
        0x143C, 0x3C40,  0x09B8, 0x3880,  0x05C0, 0x7000,  0x00CF, 0x0400,
        0x000C, 0x6000,
        },
    /*----- bitmap Boing, frame 5:  w = 32, h = 25 ------ */
        {
        0x0026, 0x2000,  0x00FC, 0x7400,  0x0187, 0x7200,  0x030F, 0x0100,
        0x0E0F, 0x0E80,  0x319F, 0x0E00,  0x23C6, 0x0F30,  0x63C1, 0xCF30,
        0x4781, 0xF310,  0x0783, 0xE0D0,  0x7383, 0xE0E0,  0x70C3, 0xE0E0,
        0x70F9, 0xE1E0,  0x70F8, 0x21E0,  0x70F8, 0x3FE0,  0x91F0, 0x3E38,
        0x4FF0, 0x7C30,  0x4E10, 0x7C60,  0x4E0F, 0x7860,  0x2E1F, 0x08C0,
        0x0E1E, 0x0E00,  0x049E, 0x1C80,  0x00E4, 0x3800,  0x00C7, 0x9000,
        0x000E, 0x6000,
        }
    };

/* these structures contain the initialization data for the animation
** sequence.
*/
NEWBOB newBoingBob =
    {
    NULL, BOING_WWIDTH, BOING_HEIGHT, BOING_DEPTH, 0x2, 0x0,
    SAVEBACK | OVERLAY, 0, RBMDEPTH, 0,0,0,0,
    };
NEWANIMSEQ newBoingSeq =
    {
    NULL, (WORD *)boing3Image, boing3XTranses, boing3YTranses,
    boing3Times, boing3CRoutines, 0, BOING_COUNT, 0,
    };

/*---------------------------------------------------------------------------
** setupBoing()
**
** make a new animation object.  since all of the boing balls use the same
** underlying data, the initalization structures are hard-coded into the
** routine (newBoingBob and newBoingSeq.)
**
** return a pointer to the object if successful, NULL if failure.
** set a global error code on failure.
*/
struct AnimOb *setupBoing(SHORT dbufing)
{
struct AnimOb   *bngOb;
struct AnimComp *bngComp;

if (NULL != (bngOb = AllocMem((LONG)sizeof(struct AnimOb), MEMF_CLEAR)))
    {
    newBoingBob.nb_DBuf    = dbufing;     /* double-buffer status  */
```

```
    newBoingSeq.nas_HeadOb = bngOb;        /* pass down head object */

    if (NULL != (bngComp = makeSeq(&newBoingBob, &newBoingSeq)))
        {
        bngOb->HeadComp = bngComp;  /* the head comp is the one that */
                                    /* is returned by makeSeq()      */
        return(bngOb);
        }
    FreeMem(bngOb, (LONG)sizeof(struct AnimOb));
    }
return_code = RETURN_WARN;
return(NULL);
}

/*---------------------------------------------------------------------------
** runAnimation()
**
** a simple message handling loop that also draws the successive frames.
*/
VOID runAnimation(struct Window *win, SHORT dbufing,
                  struct AnimOb **animKey, struct BitMap **myBitMaps)
{
struct IntuiMessage  *intuiMsg;
WORD toggleFrame;

/* toggleFrame is used to keep track of which frame of the double buffered
** screen we are currently displaying.  the variable must exist for the life
** of the displayed objects, so it is defined here.
*/
toggleFrame = 0;

/* end the loop on a CLOSEWINDOW event */
for (;;)
    {
    /* draw the gels, then check for messages.
    ** check the messages after each display so we get a quick response.
    */
    drawGels(win, animKey, dbufing, &toggleFrame, myBitMaps);

    /* quit on a control_c */
    if (SetSignal(0L,0L) & SIGBREAKF_CTRL_C)
        return;

    /* check for a closewindow event, die if found */
    while (intuiMsg = (struct IntuiMessage *)GetMsg(win->UserPort))
        {
        if (intuiMsg->Class == CLOSEWINDOW)
            {
            ReplyMsg((struct Message *)intuiMsg);
            return;
            }
        ReplyMsg((struct Message *)intuiMsg);
        }
    }
}

/*---------------------------------------------------------------------------
** setupPlanes()
**
** called only for double-buffered displays.
** allocate and clear each bit-plane in a bitmap structure.
** clean-up on failure.
*/
LONG setupPlanes(struct BitMap *bitMap, LONG depth, LONG width, LONG height)
```

```
{
SHORT plane_num ;

for (plane_num = 0; plane_num < depth; plane_num++)
    {
    if (NULL != (bitMap->Planes[plane_num] =
                    (PLANEPTR)AllocRaster(width, height)))
        BltClear((APTR)(bitMap->Planes[plane_num]), (width / 8) * height, 1);
    else
        {
        freePlanes(bitMap, depth, width, height);
        return_code = RETURN_WARN;
        return(NULL);
        }
    }
return(TRUE);
}

/*-----------------------------------------------------------------------
** setupBitMaps()
**
** allocate the two bitmaps for a double-buffered display.
** routine only called when the display is double-buffered.
*/
struct BitMap **setupBitMaps(LONG depth, LONG width, LONG height)
{
static struct BitMap *myBitMaps[2];

/* allocate the two bit-map structures. these do not have to be in CHIP */
if (NULL != (myBitMaps[0] =
        (struct BitMap *)AllocMem((LONG)sizeof(struct BitMap), MEMF_CLEAR)))
    {
    if (NULL != (myBitMaps[1] =
        (struct BitMap *)AllocMem((LONG)sizeof(struct BitMap), MEMF_CLEAR)))
        {
        /* initialize the bit maps to the correct size. */
        InitBitMap(myBitMaps[0], (BYTE)depth, (SHORT)width, (SHORT)height);
        InitBitMap(myBitMaps[1], (BYTE)depth, (SHORT)width, (SHORT)height);

        /* allocate and initialize the bit-planes for the bit-maps. */
        if (NULL != setupPlanes(myBitMaps[0], depth, width, height))
            {
            if (NULL != setupPlanes(myBitMaps[1], depth, width, height))
                return(myBitMaps);

            freePlanes(myBitMaps[0], depth, width, height);
            }
        FreeMem(myBitMaps[1], (LONG)sizeof(struct BitMap));
        }
    FreeMem(myBitMaps[0], (LONG)sizeof(struct BitMap));
    }
/* on failure, everything is freed and a global return code is set. */
return_code = RETURN_WARN;
return(NULL);
}

/*-----------------------------------------------------------------------
** freePlanes()
**
** free all of the bit-planes in a bit-map structure.
*/
VOID freePlanes(struct BitMap *bitMap, LONG depth, LONG width, LONG height)
{
SHORT plane_num ;
```

```
for (plane_num = 0; plane_num < depth; plane_num++)
    {
    if (NULL != bitMap->Planes[plane_num])
        FreeRaster(bitMap->Planes[plane_num], (USHORT)width, (USHORT)height);
    }
}

/*------------------------------------------------------------------------
** freeBitMaps()
**
** free the two bit-maps from the double buffered display.
** the bit-planes are freed first, then the bit-map structures.
*/
VOID freeBitMaps(struct BitMap **myBitMaps, LONG depth, LONG width, LONG height)
{
freePlanes(myBitMaps[0], depth, width, height);
freePlanes(myBitMaps[1], depth, width, height);

FreeMem(myBitMaps[0], (LONG)sizeof(struct BitMap));
FreeMem(myBitMaps[1], (LONG)sizeof(struct BitMap));
}

/*------------------------------------------------------------------------
** setupDisplay()
**
** open the screen and the window for the display.
**
** if using double buffered display, assume the bit-maps have been opened
** and correctly set-up.
*/
struct GelsInfo *setupDisplay(struct Window **win, SHORT dbufing,
                              struct BitMap **myBitMaps)
{
struct GelsInfo    *gInfo;
struct Screen      *screen;

/* if double-buffered, set-up the new screen structure for custom bit map */
if (dbufing)
    {
    ns.Type |= CUSTOMBITMAP;
    ns.CustomBitMap = myBitMaps[0];
    }

/* open everything.  check for failure. */
if ((screen = (struct Screen *)OpenScreen(&ns)) != NULL)
    {
    nw.Screen = screen;
    if ((*win = (struct Window *)OpenWindow(&nw)) != NULL)
        {
        if (dbufing)
            {
            /* we are double buffered.  set the rastport for it. */
            (*win)->WScreen->RastPort.Flags = DBUFFER;

            /* this copies the intuition display (close gadget) to the
            ** second bit-map so the display does not flash when we change
            ** between them.
            */
            (*win)->WScreen->RastPort.BitMap = myBitMaps[1];
            BltBitMapRastPort(myBitMaps[0], 0,0, &(*win)->WScreen->RastPort,
                        0,0, RBMWIDTH, RBMHEIGHT, 0xC0);
            (*win)->WScreen->RastPort.BitMap = myBitMaps[0];
            }
```

```
        /* ready the gel system for accepting objects
        ** this is only done once for each rastport in use.
        */
        if (NULL != (gInfo = setupGelSys(&(*win)->WScreen->RastPort, 0xFC)))
            return(gInfo);

        CloseWindow(*win);
        }
    CloseScreen(screen);
    }
/* on an error everything is cleaned-up, and a global return code is set. */
return_code = RETURN_WARN;
return(NULL);
}

/*-------------------------------------------------------------------------
** drawGels()
**
** handle the update of the display.  Animate the simulation and check for
** collisions.  If the screen is double buffered, swap the bit map as
** required.
*/
VOID drawGels(struct Window *win, struct AnimOb **animKey, SHORT dbufing,
              WORD *toggleFrame, struct BitMap **myBitMaps)
{
/* do the required animation stuff.  you must sort both after the animate
** call and after the collision call.
*/
Animate((struct AnimOb *)animKey, &win->WScreen->RastPort);
SortGList(&win->WScreen->RastPort);

DoCollision(&win->WScreen->RastPort);
SortGList(&win->WScreen->RastPort);

/* toggle if double buffered */
if (dbufing)
    win->WScreen->ViewPort.RasInfo->BitMap = myBitMaps[*toggleFrame];

/* draw the new position of the gels into the screen. */
DrawGList(&win->WScreen->RastPort, &win->WScreen->ViewPort);

/* if using a double buffered display, you have a more complicated
** update procedure.
**
** if not, then simply use WaitTOF().
*/
if (dbufing)
    {
    MakeScreen(win->WScreen);
    RethinkDisplay();

    *toggleFrame ^= 1;
    win->WScreen->RastPort.BitMap = myBitMaps[*toggleFrame];
    }
else
    WaitTOF();
}

/*-------------------------------------------------------------------------
** bounceWall()
**
** handle bouncing the animation objects off of the walls.
**
```

```
** use __stdargs and __saveds because this routine is not directly called
** by this program.  The call to DoCollision() causes a call back to
** this routine when an animation object comes in contact with a wall.
** __stdargs says the arguments are passed on the stack.
** __saveds says to restore the data segment pointer on entry to the routine.
*/
VOID __stdargs __saveds bounceWall(struct VSprite *vs1,LONG borderflags)
{
struct AnimOb *ob;

/* get a pointer to the object from the sprite pointer. */
ob = vs1->VSBob->BobComp->HeadOb;

/* check for hits and act apropriately.
** for right and left, reverse the x velocity if the object is moving
** towards the wall (it may have already reversed but still be in contact
** with the wall.)
** for the bottom and top you also have to subtract out the acceleration.
*/
if (((borderflags & RIGHTHIT) && (ob->XVel > 0)) ||
    ((borderflags & LEFTHIT) && (ob->XVel < 0)))
    ob->XVel = -(ob->XVel);
else if (((borderflags & TOPHIT) && (ob->YVel < 0)) ||
         ((borderflags & BOTTOMHIT) && (ob->YVel > 0)))
    {
    ob->YVel -= ob->YAccel;
    ob->YVel = -(ob->YVel);
    }
}

/*---------------------------------------------------------------------------
** hit_routine()
**
** handle the collision between two animation objects.
** this routine simulates objects bouncing off of each other.
** this does not do a very good job of it, it does not take into account
** the angle of the collision or real physics.
** if anyone wants to fix it, please feel free.
**
** use __stdargs and __saveds because this routine is not directly called
** by this program.  The call to DoCollision() causes a call back to
** this routine when two animation objects overlap.
** __stdargs says the arguments are passed on the stack.
** __saveds says to restore the data segment pointer on entry to the routine.
*/
VOID __stdargs __saveds hit_routine(struct VSprite *vs1,struct VSprite *vs2)
{
LONG vel1, vel2;

/* check that the bob is not being removed!  This is due to a 1.3 bug
** where all bobs are tested for collision, even the ones that are in
** the process of being removed.  See text for more information.
**
** bobs are moved to a very large negative position as they are being
** removed.  if the BOBSAWAY flag is set, then both bobs in the collision
** are in the process of being removed--don't do anything in the collision
** routine.
*/
if (!(vs1->VSBob->Flags & BOBSAWAY))
    {
    /* cache the velocity values
    ** Do the X values first (order is not important.)
    */
    vel1 = vs1->VSBob->BobComp->HeadOb->XVel;
```

```
    vel2 = vs2->VSBob->BobComp->HeadOb->XVel;

    /* if the two objects are moving in the opposite direction (X component)
    ** then negate the velocities.
    ** else swap the velocities.
    */
    if (((vel1 > 0) && (vel2 < 0)) || ((vel1 < 0) && (vel2 > 0)))
        {
        vs1->VSBob->BobComp->HeadOb->XVel = -vel1;
        vs2->VSBob->BobComp->HeadOb->XVel = -vel2;
        }
    else
        {
        vs1->VSBob->BobComp->HeadOb->XVel = vel2;
        vs2->VSBob->BobComp->HeadOb->XVel = vel1;
        }

    /* cache the velocity values
    ** Do the Y values second (order is not important.)
    */
    vel1 = vs1->VSBob->BobComp->HeadOb->YVel;
    vel2 = vs2->VSBob->BobComp->HeadOb->YVel;

    /* if the two objects are moving in the opposite direction (Y component)
    ** then negate the velocities.
    ** else swap the velocities.
    */
    if (((vel1 > 0) && (vel2 < 0)) || ((vel1 < 0) && (vel2 > 0)))
        {
        vs1->VSBob->BobComp->HeadOb->YVel = -vel1;
        vs2->VSBob->BobComp->HeadOb->YVel = -vel2;
        }
    else
        {
        vs1->VSBob->BobComp->HeadOb->YVel = vel2;
        vs2->VSBob->BobComp->HeadOb->YVel = vel1;
        }
    }
}

/*------------------------------------------------------------------------
** main routine
**
** run a double buffered display if the user puts any arguments on the
** command line.
**
** open libraries, set-up the display, set-up the animation system and
** the objects, set-up collisions between objects and against walls,
** and run the thing.
**
** clean-up and close resources when done.
*/
VOID main(int argc, char **argv)
{
struct BitMap   **myBitMaps;
struct AnimOb    *boingOb;
struct AnimOb    *boing2Ob;
struct AnimOb    *boing3Ob;
struct Window    *win;
struct Screen    *screen;
struct GelsInfo  *gInfo;
struct AnimOb    *animKey;
SHORT dbufing;
```

```
return_code = RETURN_OK;

/* if any arguments, use double-buffering */
if (argc > 1)
    dbufing = 1;
else
    dbufing = 0;

/* open required libraries */
if (NULL == (IntuitionBase = (struct IntuitionBase *)
        OpenLibrary("intuition.library", 33L)))
    return_code = RETURN_FAIL;
else
    {
    if (NULL == (GfxBase = (struct GfxBase *)
            OpenLibrary("graphics.library", 33L)))
        return_code = RETURN_FAIL;
    else
        {
        /* note that setupBitMaps() will only be called if
        ** we are double buffering
        */
        if ((!dbufing) ||
            (NULL != (myBitMaps=setupBitMaps(RBMDEPTH,RBMWIDTH,RBMHEIGHT))))
            {
            if (NULL != (gInfo = setupDisplay(&win,dbufing,myBitMaps)))
                {
                /* you have to initialize the animation key
                ** before you use it.
                */
                InitAnimate(&animKey);

                /* set-up the first boing ball.
                ** all of these use the same data, hard coded into setupBoing().
                ** change the color by changing PlanePick.
                ** set the ID of the ball (MeMask) to BOING_1.
                ** HitMask = 0xFF means that it will collide with everything.
                 */
                newBoingBob.nb_PlanePick = 0x2;
                newBoingBob.nb_MeMask    = 1L<<BOING_1;
                newBoingBob.nb_HitMask   = 0xFF;
                if (NULL != (boingOb = setupBoing(dbufing)))
                    {
                    /* pick an initial position, velocity and acceleration
                    ** and add the object to the system.  NOTE that the
                    ** Y-velocity and X-acceleration are not set (they default
                    ** to zero.)  This means that the objects will maintain
                    ** a constant movement to the left or right, and will
                    ** rely on the Y accelleration for the downward movement.
                    ** The collision routines change these values, producing
                    ** bouncing off of walls and other objects.
                    **
                    ** NOTE:  ANFRACSIZE is a value that shifts animation
                    ** constants past an internal decimal point.  If you
                    ** do not do this, then the values will only be some
                    ** fraction of what you expect.  See the Rom Kernel
                    ** Manual:  Libraries and Devices.
                    */
                    boingOb->AnY    = 10 << ANFRACSIZE;
                    boingOb->AnX    = 250 << ANFRACSIZE;
                    boingOb->XVel   = -(4 << ANFRACSIZE);
                    boingOb->YAccel = 70;
                    AddAnimOb(boingOb, (APTR)&animKey,
                              &win->WScreen->RastPort);
```

```
            /* do the second object--see above comments. */
            newBoingBob.nb_PlanePick = 0x1;
            newBoingBob.nb_MeMask    = 1L<<BOING_2;
            newBoingBob.nb_HitMask   = 0xFF;
            if (NULL != (boing2Ob = setupBoing(dbufing)))
                {
                boing2Ob->AnY    = 50 << ANFRACSIZE;
                boing2Ob->AnX    = 50 << ANFRACSIZE;
                boing2Ob->XVel   = 3 << ANFRACSIZE;
                boing2Ob->YAccel = 70;
                AddAnimOb(boing2Ob, (APTR)&animKey,
                        &win->WScreen->RastPort);

                /* do the third object--see above comments.
                ** here we also use PlaneOnOff to change the color.
                */
                newBoingBob.nb_PlanePick = 0x1;
                newBoingBob.nb_PlaneOnOff= 0x2;
                newBoingBob.nb_MeMask    = 1L<<BOING_3;
                newBoingBob.nb_HitMask   = 0xFF;
                if (NULL != (boing3Ob = setupBoing(dbufing)))
                    {
                    boing3Ob->AnY    = 80 << ANFRACSIZE;
                    boing3Ob->AnX    = 150<< ANFRACSIZE;
                    boing3Ob->XVel   = 2 << ANFRACSIZE;
                    boing3Ob->YAccel = 70;
                    AddAnimOb(boing3Ob, (APTR)&animKey,
                            &win->WScreen->RastPort);

                    /* set up the collisions between boing balls.
                    ** NOTE that they all call the same routine.
                    */
                    SetCollision(BOING_1,hit_routine,gInfo);
                    SetCollision(BOING_2,hit_routine,gInfo);
                    SetCollision(BOING_3,hit_routine,gInfo);

                    /* set the collisions with the walls. */
                    SetCollision(BORDERHIT,bounceWall,gInfo);

                    /* everything set-up...run the animation. */
                    runAnimation(win, dbufing, &animKey, myBitMaps);

                    /* done...
                    ** free-up everything and clean up the mess.
                     */
                    freeOb(boing3Ob, RBMDEPTH);
                    }
                freeOb(boing2Ob, RBMDEPTH);
                }
            freeOb(boingOb, RBMDEPTH);
            }

        cleanupGelSys(gInfo, &win->WScreen->RastPort);
        screen = win->WScreen;
        CloseWindow(win);
        CloseScreen(screen);
        }

    if (dbufing)
        freeBitMaps(myBitMaps, RBMDEPTH, RBMWIDTH, RBMHEIGHT);
    }
CloseLibrary((struct Library *)GfxBase);
}
```

```
    CloseLibrary((struct Library *)IntuitionBase);
    }

/* return the global return code to the system. */
exit(return_code);
}
```

Stopwatch Functions with the timer.device

by Mike Sinz

To measure time on the Amiga the system software provides the *timer.device*. The *timer.device* is very handy for many purposes, however, it does not have a built-in stopwatch. It does have some time manipulation functions that make this a simple thing to do though. This article shows how to do stopwatch functions using the *timer.device*.

The example code contains four routines needed to set up and use the stopwatch. The routines are:

```
struct timerequest  *Init_Timer  (VOID)
VOID  Timer_Start (struct timerequest *)
VOID  Timer_Stop  (struct timerequest *)
VOID  Free_Timer  (struct timerequest *)
```

The stopwatch functions all use a struct timerequest as their main parameter. This is just the usual Amiga IORequest structure with two extra fields for seconds and microseconds. Init_Timer() sets up the *timer.device* and returns a pointer to a struct timerequest. This pointer is used in Timer_Start() and Timer_Stop() which measure the elapsed time. Finally, Free_Timer() is used to close the *timer.device* and free the memory used for the timerequest.

An example main() function is listed below to show how to use the stopwatch functions to measure elapsed time. The example just does a Delay(73L) and then displays the amount of time that took. Some uses for the program might include measuring how long a student takes to answer a question, the amount of time a player has been playing, or how fast the system is. For more information on the *timer.device*, refer to the *ROM Kernel Manual: Libraries and Devices* (p. 871, ISBN 0-201-18187-8).

```
/*
 * Example stopwatch functions using the Amiga timer.device...
 */

/*
 * Makefile used to compile with Lattice C 5.04 or 5.05
 *

#
# MakeFile for StopWatch
#

CFLAGS= -b1 -cfirstq -ms0 -rr1 -v -w

OBJS=   StopWatch.o

LIBS=   LIB:lcsr.lib

.c.o:
@LC $(CFLAGS) $*

StopWatch: $(OBJS)
@BLink FROM LIB:c.o $(OBJS) TO StopWatch LIB $(LIBS) SMALLDATA SMALLCODE
 *
 *
 */

#include    <exec/types.h>
#include    <exec/memory.h>
#include    <devices/timer.h>

#include    <proto/exec.h>
#include    <proto/timer.h>
#include    <proto/dos.h>

#include    <stdio.h>

/*
 * The library base for the timer...
 */
struct  Library *TimerBase=NULL;

/*
 * This gets the starting time...
 */
VOID Timer_Start(struct timerequest *Time_Req)
{
    Time_Req->tr_node.io_Command=TR_GETSYSTIME;
    Time_Req->tr_node.io_Flags=IOF_QUICK;
    DoIO((struct IORequest *)Time_Req);
}
```

```
/*
 * This gets the ending time and computes the time difference
 * in the timerequest->tr_time.
 */
VOID Timer_Stop(struct timerequest *Time_Req)
{
struct  timeval StartTime;

    StartTime=Time_Req->tr_time;

    Time_Req->tr_node.io_Command=TR_GETSYSTIME;
    Time_Req->tr_node.io_Flags=IOF_QUICK;
    DoIO((struct IORequest *)Time_Req);

    SubTime(&(Time_Req->tr_time),&StartTime);
}

/*
 * Initialize the stopwatch...
 */
struct timerequest *Init_Timer(VOID)
{
register    struct  timerequest *Time_Req=NULL;
register    struct  MsgPort     *port=NULL;


    if (port=CreatePort(NULL,NULL))
    {
        if (Time_Req=(struct timerequest *)CreateExtIO(port,
                                            sizeof(struct timerequest)))
        {
            if (!OpenDevice(TIMERNAME,UNIT_VBLANK,
                            (struct IORequest *)Time_Req,NULL))
            {
                TimerBase=(struct Library *)Time_Req->tr_node.io_Device;
            }
            else
            {
                DeleteExtIO((struct IORequest *)Time_Req);
                Time_Req=NULL;
            }
        }
        if (!Time_Req)
        {
            DeletePort(port);
            port=NULL;
        }
    }

    return(Time_Req);
}
```

```
/*
 * Free up the timer...
 */
VOID Free_Timer(struct timerequest *Time_Req)
{
    if (Time_Req)
    {
        CloseDevice((struct IORequest *)Time_Req);
        DeletePort(Time_Req->tr_node.io_Message.mn_ReplyPort);
        DeleteExtIO((struct IORequest *)Time_Req);
    }
}


/*
 * A simple main() to use these features...
 */
VOID main(int argc, char *argv[])
{
register    struct  timerequest *StopWatch;

    if (argc)   /* Check if CLI... */
    {
        if (StopWatch=Init_Timer())
        {
            Timer_Start(StopWatch);

            /* Now we do something... */
            Delay(73L);

            Timer_Stop(StopWatch);

            /* Now we display the results... */

            printf("Time taken: %ld.%06ld seconds\n",
                        StopWatch->tr_time.tv_secs,
                        StopWatch->tr_time.tv_micro);

            /* Free the structure again... */
            Free_Timer(StopWatch);
        }
    }
}
```

Amiga Mail

May/June 1990

From the Desk of Jeff Scherb

It's DevCon time again, and it is worth taking the time here to reinforce the theme of this year's conference - "Moving Up to 2.0." Release 2.0 is a significant improvement in both the appearance and the performance of the Amiga OS. Many of these new features can directly help you to build better applications. In addition, the new release will advance Commodore's market strategy for the Amiga - multimedia in the corporate environment.

Release 2.0 gives the Amiga a new look. The operating system now contains state-of-the-art imagery and easy-to-use features which make the machine much more appealing to corporate markets. If your products are aimed at this market, you should consider revising them to take advantage of the new features in 2.0.

Commodore realizes the importance of maintaining a consistent look and feel among all applications. This makes each new application easier to use, easier to learn, and therefore easier to sell. It also helps the Amiga gain acceptance in the growing professional market. Release 2.0 can help achieve this goal. By using new features such as the gadtools.library and system-supplied requestors, you can assure consistent operation in your applications. Commodore is also working to help you with an official manual on User Interface Guidelines. Until that document is published, the new OS and AmigaVision can be used as prime examples of the standards we advocate.

Release 2.0 provides more than just an improved appearance, it can also give new functionality to your applications. With the inclusion of ARexx, release 2.0 brings a standard method of simple inter-process communication to every Amiga user. With ARexx, you can easily add scripting capabilities to your programs or provide other custom features to the user that were never before possible.

Developers with professional and productivity applications aren't the only ones who can benefit from release 2.0. The new video modes and resolutions, for example, can provide new functionality for games and entertainment software, reducing the need to resort to creative hardware hacks.

In summary, we at Commodore think that release 2.0 offers new opportunities for improvement in Amiga application software, expanding the market and revenue for you and Commodore alike. Since many of the new features in 2.0 were requested by you at past conferences, it's time to put your great ideas to work! I know you will enjoy and benefit from the 2.0 sessions this year in Atlanta. Now that 2.0 is a reality, let's all work together to make it a success.

JS

REVISIONS

The following are revised pages from past *Amiga Mail* articles. Simply substitute these pages for the corresponding ones already in your *Amiga Mail* binder. Please take the time to do this now so your binder contains the most up-to-date information available.

CONTENTS

Amiga Mail

May/June 1990

I - Programming Practices and Compatability

Official Warning to ROM-Jumpers, Structure-Hackers, and Others 1
from Commodore Engineering, Commodore-Amiga and CATS

Avoiding Port, Task, and File Name Conflicts 3
Carolyn Scheppner

Writing Software For All the Amigas 5
Dan Baker

PAL and NTSC Differences 9
Carolyn Scheppner

Testing For European Compatability Without Going to Europe 11
Dan Baker

Traveling with Your Amiga 13
Dan Baker and Don Gilbreath

Getting Software to Run on Hard Disks 15
Michael Colligon

Five Alive (Or How to Help Your Program Survive) 19
Dan Schein

Programming For Portability and Compatibility 21
Bill Koester and Phil Lindsay

PAL-NTSC Switch for the A2000 31
Hedley Davis

Troubleshooting Your Software 35
Carolyn Scheppner

II - Amiga DOS

Fast Filing System 1
Steve Beats

Tricks with AmigaDOS, Part II 5
Andy Finkel

New Packets and Structures in 1.2 AmigaDOS 13
Carolyn Scheppner

Low Level Differences Between FFS and the ROM Filing System 21
Steve Beats

WHICH.C - Locates your commands 31
Carolyn Scheppner

Infodata 37
Bryce Nesbitt

Setting Up Your Hard Disk for Workbench V1.3 Part I 39
Bill Koester

Setting Up Your Hard Disk for WorkBench V1.3 Part II 47
Bill Koester

CON: and Raw Event Streams 59
Carolyn Scheppner

CLI Colors.asm: Using Console Packets in Assembler 65
Michael Sinz

How to Find the Stack Size at Run Time 73
Bryce Nesbitt

Creating Multiple Processes with Re-entrant Code 77
Michael Sinz

Using C to Get a Pointer the CLI Window 93
Rob Wyesham

Amiga Mail

III - Exec

Exception Vector Warning 1
Bryce Nesbitt
Assembler Device IO - The Easy Way 3
Carolyn Scheppner
Software Reboot Revisited 9
Bryce Nesbitt
Creating Your Own Libraries 11
Michael Sinz
Identifying the Amiga's CPU 35
Dave Haynie

IV - Intuition and Workbench

Opening Full Size Screens and Windows On Any System 1
Carolyn Scheppner - CATS
Disabled Render - How to Supply an Alternate Image for a Disabled Gadget 9
Adam Keith Levin Commodore-Amiga Technical Support
Mutual Exclusion of Gadgets 23
Peter Chema
Paced Repeat Key 37
Michael Sinz
Modular Event Processing 41
David Junod

V - Graphics

Creating Sprites in C from the Intuition Pointer 1
Adam Levin
Extended Text Fonts 9
Andy Finkel
The Lost Blitter Docs 13
Carolyn Scheppner
Disabling Sprite DMA From a Vertical Blank Interrupt 17
Adam Kieth Levin
Amiga Monitors and Display Modes 23
Staff
Collision Detection Between Animation Objects 25
Ken Farinsky

VI - Audio

PAL and Audio 1
Carolyn Scheppner
Simple Audio 3
Dan Baker
DTMF - A Program for Generating Touch-Tone Signals 7
Adam Levin
Amiga Audio Cutoff Filter 23
Dan Schein
DebTones.asm - A Low Level Debugging Macro 27
Carolyn Scheppner
Reading Macintosh Sound Files 29
John Orr

VII - IFF

Intro to IFF Amiga ILBM Files and Amiga Viewmodes 1
Carolyn Scheppner
A SMUS Player 7
Dan Baker
8SVX: Playing Samples Larger Than 128K 33
Dan Baker
Default Bit Ordering in the Storage of Deep ILBMs 43
Carolyn Scheppner

VIII - Trackdisk

Notes on 1.2 Amiga Trackdisk Format 1
Carolyn Scheppner and Andy Finkel
Amiga Low-Level Disk Access 9
Bryce Nesbitt
Amiga Floppy Drives 15
Bryce Nesbitt
More on Low-Level Disk Access 19
Randell Jesup

IX - Timers, Serial Port and Parallel Port

Using the Serial and Parallel Port Resources 1
Dan Baker and Bryce Nesbitt
A Timer Using the CIA Resource 5
Adam Levin and Paul Higginbottom
Amiga Multiple Serial Ports 15
Bryce Nesbitt
How To Waste Time 21
Bryce Nesbitt
SoftInt MsgPorts and the Timer Device 25
Carolyn Scheppner
MIDI Output in Assembler 33
Darius Taghavy
Stopwatch Functions with the timer.device 37
Mike Sinz

X - Printers

PostScript Primer for Programmers 1
Carolyn Scheppner
V1.3 Preferences 9
Eric Cotton and David Berezowski
Programmer Notes on Calling the V1.3 Printer Device 23
David Berezowski
Changes to the Printer Device for V1.3 33
David Berezowski
Troubleshooting PostScript Printers 37
Bryce Nesbitt
Writng a V1.3 Graphics Printer Driver 43
David Berezowski
Printer.c - A Printer Text and Graphics Module 57
Carolyn Scheppner

Amiga Mail

XI - Input Devices

International Keyboard Input 1
Eric Cotton and Carolyn Scheppner
Allocation of Input and Other System Resources 7
Bob Burns
Amiga Keyboard Input and Keymaps 23
Bryce Nesbitt
Joystick Port Output 29
Carolyn Scheppner

XII - Expansion, Autoconfig and Autoboot

Using Byte-Wide Autoboot ROMs Under Amiga OS V1.3 1
Steve Beats
Using the IO Space In Time Critical Applications 3
Robert V. Welland
V1.3 Kickstart and Autoboot 7
Bart Whitebook
Your Manufacturer Number 19
Dan Baker
68030 Hardware Design Note 25
George Robbins
PAL and NTSC Genlock Interface Guidelines 27
Scott Hood
Expanding Fast RAM on the Amiga 3000 31
Rob Wyesham

XIII - Math

Introduction to 1.3 IEEE Double Precision Libraries 1
Dale Luck
Amiga Math: Fast Fourier Transform 11
Dan Baker

XIV - Miscellaneous Topics

The Amiga Virus 1
Bill Koester
Using the Clipboard Device 5
Andy Finkel
Stereoscopy and the Amiga 15
Tracy McSherry
Amiga 2000 Motherboard Revision History 21
Staff